Dossier technique - Projet "Spectromètre à Réseau"

Lucien Kergadallan, Rémi Truong, Gabriel Peyre–Pradat, Haoyang Li

Sommaire

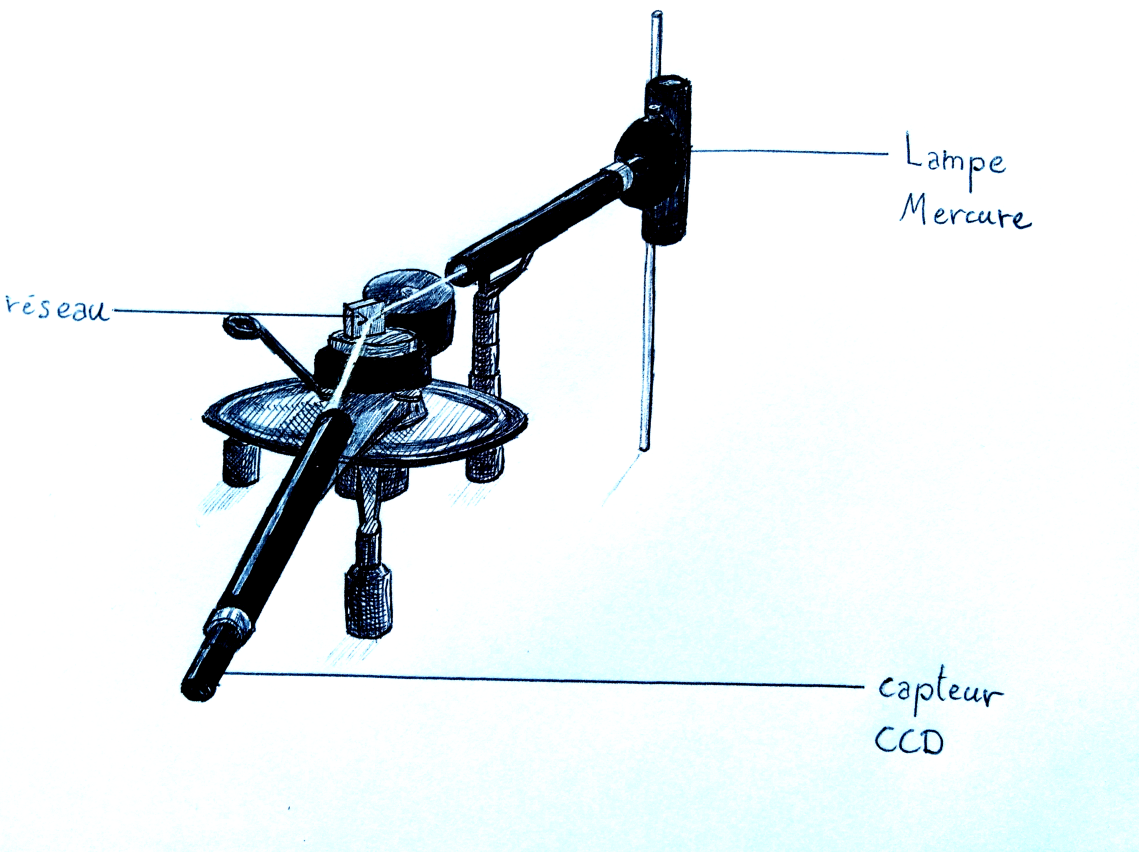


Figure 1: Goniomètre utilisé lors du projet

1 Introduction

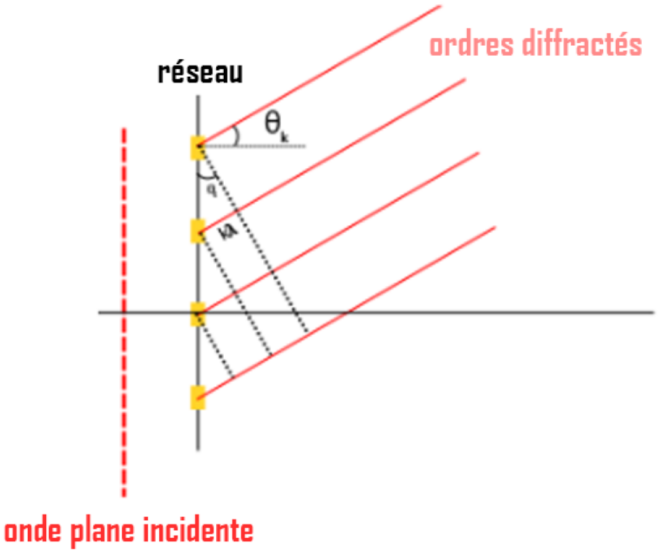


Figure 2: Schéma de principe d'un réseau en transmission

Le principe d'un spectromètre est de permettre d'observer et/ou d'étudier le spectre d'une source de lumière. Une façon de faire cela est d'utiliser un prisme ou un réseau. Dans ce projet nous avons utilisé un réseau en réflexion, de pas $a = 70$ trous par mm. Rappelons en préliminaire la formule liant l'angle avec lequel la lumière est diffractée selon l'ordre k :

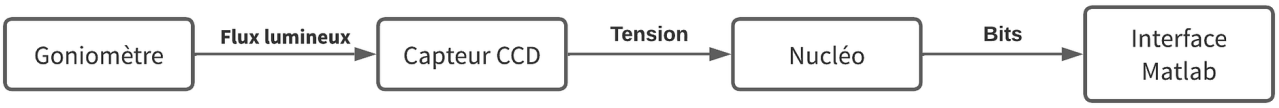
$$\theta_k = k \cdot \frac{\lambda}{a}$$

Ici, le schéma de la figure 2 est pour un réseau en transmission, mais le calcul et le principe physique sont les mêmes pour un réseau utilisé en réflexion, comme nous le faisons ici. Ainsi, la lumière en "sortie" du réseau est composée de différents ordres, chaque ordre contenant l'intégralité du spectre de la lumière incidente. On a donc rapidement des phénomènes de recouvrement d'ordres lorsqu'on regarde les ordres 2 ou plus.

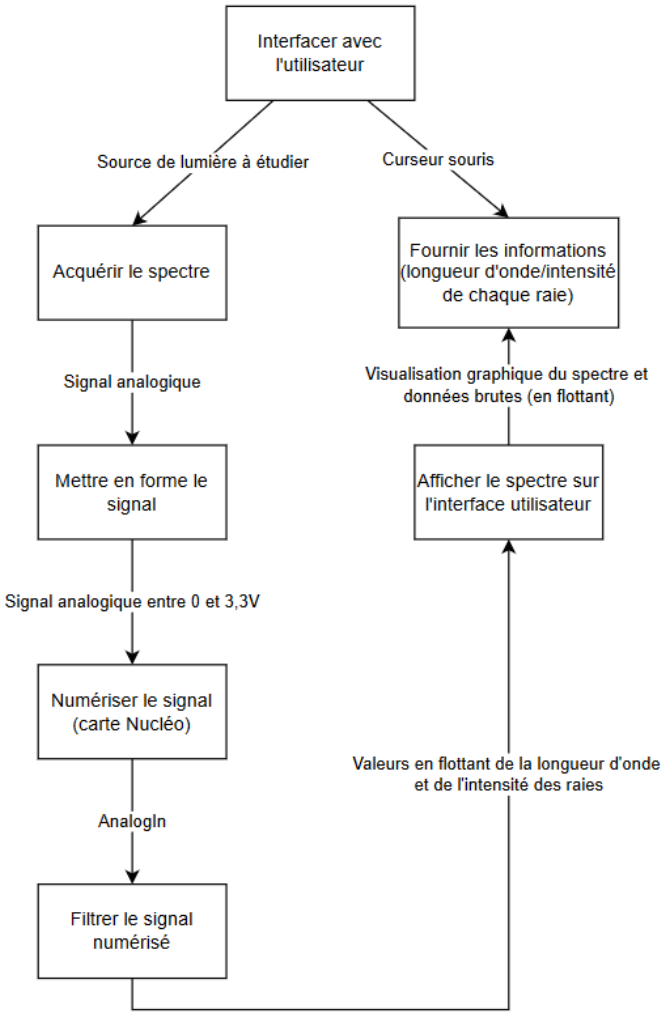
On utilise le réseau avec un goniomètre (*cf* figure 1), un appareil optique permettant d'observer le réseau sous différents angles avec un viseur, pour pouvoir balayer le spectre et les différents ordres diffractés. L'idée du projet est de placer un capteur en sortie de ce viseur, pour pouvoir mesurer le spectre, plutôt que l'observer à l'oeil.

2 Principe et vision initale du système

Une première compréhension du projet peut-être résumée dans le diagramme fonctionnel suivant :



Chaque bloc correspond à un système pouvant être étudié seul, l'enjeu du projet étant bien évidemment de les relier entre eux pour obtenir un système fonctionnel. Une étude plus poussée de ces différents blocs et des contraintes qui leur sont associées a permis de rapidement concevoir le diagramme fonctionnel suivant :



Les objectifs que nous nous sommes fixés au début du projet étaient les suivants :

- Pouvoir observer le spectre de la source étudiée sur une interface codée sur matlab, avec la possibilité d'utiliser un curseur pour accéder aux paramètres du spectre.
- Pouvoir obsever le spectre de la lampe spectrale à mercure utilisée, au moins en partie, et pouvoir distinguer les 3 raies qui le constituent.

3 Réalisation du projet

Une fois les objectifs définis et après avoir réfléchi au diagramme fonctionnel du système, nous avons pu commencer à realiser le projet, en séparant le travail en 3 blocs :

Optique (Lucien et Gabriel) : Une partie centrée sur le goniomètre et son utilisation, dont l'objectif final était d'utiliser un capteur CCD pour observer le spectre, les observations se faisant à l'oscilloscope.

Numérisation (Rémi) : Une autre partie visant à développer l'Interface Homme-Machine (IHM) i.e. la communication des données acquises par le capteur CCD entre une carte nucléo et matlab.

Interface (Haoyang) : Une partie visant à développer une interface matlab permettant de lancer l'acquisistion du spectre et d'accéder à ses paramètres grace à un curseur.

L'avancée de chaque sous-équipe était documentée à chaque fin de séance sur l'outil de gestion de projet *Basecamp*.
Les étapes clés de ce projet consistaient à relier chaque bloc entre eux, ce qui a fort heureusement posé moins de problèmes que prévu, du fait d'un bonne communication sur les entrées et sortie (signal analogique, données numériques, etc.) attendues de chaque bloc.

3.1 Observation du spectre avec un capteur CCD et utilisation du goniomètre

Lors de ce projet, un apprentissage de l'utilisation du goniomètre a été nécessaire pour obtenir des résultats satisfaisants. Ainsi on retiendra qu'il faut bien penser à aligner avec *précision* la lampe à vapeur utilisée avec le trou source servant à éclairer le goniomètre, et à placer le réseau de telle sorte à ce qu'il reçoive le plus de lumière possible de la source.

On peut observer les ordres diffractés à l'aide d'un viseur, ce qui permet de repérer leur position. L'idée ici est d'utiliser un *oculaire* à la place du viseur, afin d'imager des ordres diffractés (qui sont à l'infini) sur un capteur CCD. On utilise le capteur CCD horizontalement; on observe donc la lumière diffractée sur les 64 pixels du capteur, couvrant au total une étendue de 4.5mm

Comme le capteur mesure 64 informations à chaque instant mais ne possède qu'une seule sortie, il est obligé d'envoyer ces informations en série, et il faut donc séquencer cette dernière de manière à récupérer les informations mesurées par tous les pixels du capteur. Pour cela, le capteur CCD a besoin de deux signaux de référence: un signal **pulse** et un signal **clock** (ou horloge). Leur rôle se comprend grâce à la figure ci-dessous, extraite de la documentation technique du capteur :

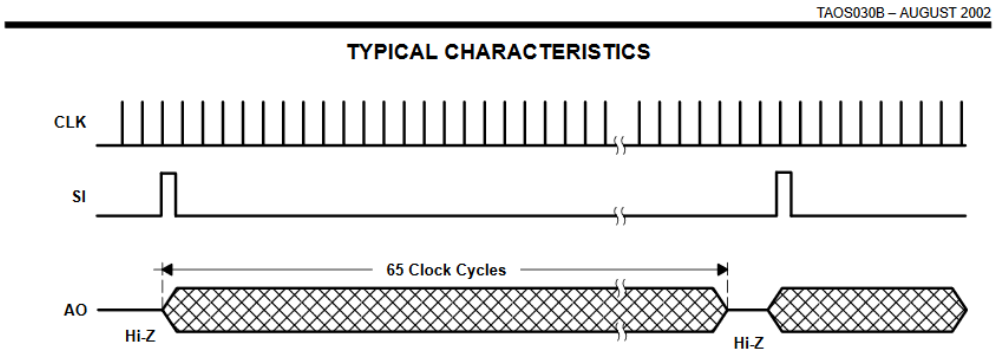


Figure 1. Timing Waveforms

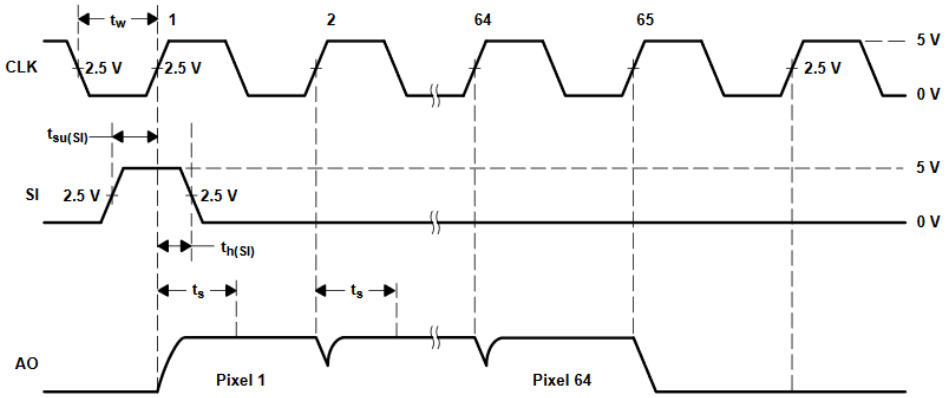


Figure 2. Operational Waveforms

Le signal **pulse** permet d'initier les séquences d'envoi des informations des pixels et le signal **clock** définit la fréquence d'envoi des informations de chaque pixel. On précise que les informations de chaque pixel sont envoyées une par une successivement. On peut jouer sur la fréquence d'envoi des pixels pour régler le temps d'intégration du capteur, et donc sa sensibilité, mais il faut quand même s'assurer que la fréquence du signal clock est *au moins* 65× plus élevée que celle du signal pulse, puisque sa période doit être *au moins* 65× moins élevée. Nous avons décidé de choisir comme paramètres $f_{pulse} = 2$ Hz et $f_{clock} = 130$ Hz.

Ces paramètres permettent d'avoir un temps d'intégration suffisament grand pour observer un signal suffisamment intense en sortie du capteur.

3.2 Communication entre la carte nucléo et matlab

Cette partie était critique en ce sens où elle nécessitait de bien comprendre les signaux obtenus par la partie Optique et d'établir un protocole de communication qui permettrait d'avoir des données numériques sous forme de *flottants* manipulables par Matlab pour la partie Interface.

Ainsi, nous avons écrit un code sur *MBED* (i.e. du C++ embarqué, proche du C) pour que la carte Nucléo remplisse élément par élément un tableau de 64 *double* dès qu'elle détecte qu'un signal de **clock** a été envoyé au capteur. Dès que le tableau est rempli, il est envoyé par un **port série** à l'ordinateur. Une fois que la carte aura détecté un signal **pulse**, elle réinitialise le tableau et le même procédé continue.

De plus, en vue de simplifier et miniaturiser le montage, la carte Nucléo génère elle-même les signaux de pulse et d'horloge, plutôt que d'utiliser des GBF externes. La subroutine du système est alors la suivante :

1. Lancement de la mesure sur Matlab
2. Matlab écoute la carte sur le port série
3. La carte envoie un Serial Input et un Clock au capteur CCD (AnalogOut)
4. La carte récupère ces signaux pour déclencher l'acquisition numérique (InterruptIn)
5. La carte effectue une mesure de la sortie du capteur dès qu'un signal d'horloge est détecté, et incrémente le compteur de pixels
6. Lorsque le compteur atteint 64 pixels, la carte envoie sur le port série le tableau de flottants rempli
7. Un signal SI est reçu, et la carte réinitialise le compteur de pixels

Un programme très simple sur Matlab a été écrit pour s'assurer que les données finales sont bien sous la forme attendue.

```
nucleo = serialport("COM7",115200); %Established connection with the Nucleo board
configureTerminator(nucleo, "CR/LF")
readline(nucleo) %Stops reading when '\r\n' is printed
clear nucleo
```

Le code *MBED* utilisé pour la carte nucléo est le suivant :

(Fichier source - main.cpp)

```
/* mbed Microcontroller Library
 * Copyright (c) 2019 ARM Limited
 * SPDX-License-Identifier: Apache-2.0
 */

#include "utility.h"

// Inputs and outputs configuration
AnalogIn    analog_in(A1);
PwmOut      pulse_pwm(D3);
PwmOut      clock_pwm(D5);
Serial      pc(USBTX, USBRX);

// Clock and SI signals used as external interrupters
InterruptIn clock_int(PA_11);
InterruptIn pulse_int(PA_12);

// System functions
void ISR_get_data(void);
void start_acquisition(void);
void send_pixel(void);

// Variables
char    data_received = 0;
double  voltage_measured[64];
double signal;
int     pixel_number = 0;

// Main function
int main(){
    pc.baud(115200);

    //Generating pulse signals
    pulse_pwm.period(0.6); //Pulse frequency : 2Hz
    pulse_pwm.pulsewidth_us(5000); // Duty Cycle

    //Generating clock signals
    clock_pwm.period_ms(8); // Clock frequency : 130Hz
    clock_pwm.write(0.5);

    pulse_int.rise(&start_acquisition);
    clock_int.fall(&send_pixel);

    while(1) {    }
}

// Start acquisition when SI input is detected
void start_acquisition(){
    pixel_number = 0;
}

void send_pixel(){
    voltage_measured[pixel_number] = analog_in.read()*3.3;
    pixel_number += 1;
    if (pixel_number > 63) {
        pixel_number = 0;
        printf_array(pc,voltage_measured);
    }
}
```

(Header - utility.h)

```
#include "mbed.h"

void printf_array(Serial& pc, double measures[]){

    for(int i = 0; i < 64; i++){
        pc.printf("%lf\r\n",measures[i]);
    }

}
```

3.3 L'interface Matlab

Cette dernière partie du projet consistait à obtenir une interface graphique facile à utiliser, pour afficher sur un graphique les données du capteur transmises par la carte Nucléo à Matlab.

Outre le partie graphique, la partie algorithmique reprenait le simple test effectué sur Matlab à la partie précédente, auquel une partie **moyennage** a été ajoutée pour diminuer le bruit ambiant.

L'utilisateur peut ainsi spécifier le nombre de mesures qu'il souhaite effectuer, tandis que le programme va établir la connexion avec la carte Nucléo par le port série, et construire un vecteur de 64 éléments lorsque la carte envoie les pixels.

Nous avions intialement programmé une partie de **calibration** mais celle-ci nécessitait qu'au moins deux pics du spectre soient capturés par le capteur CCD, afin de servir de raies de référence pour donner la longueur d'onde de tous les autres.

Nous n'avons finalement pas retenu cette partie car elle nécessitait à l'utilisateur final de changer le réseau en fonction de la source qu'il utilisait, offrant peu de flexibilité.

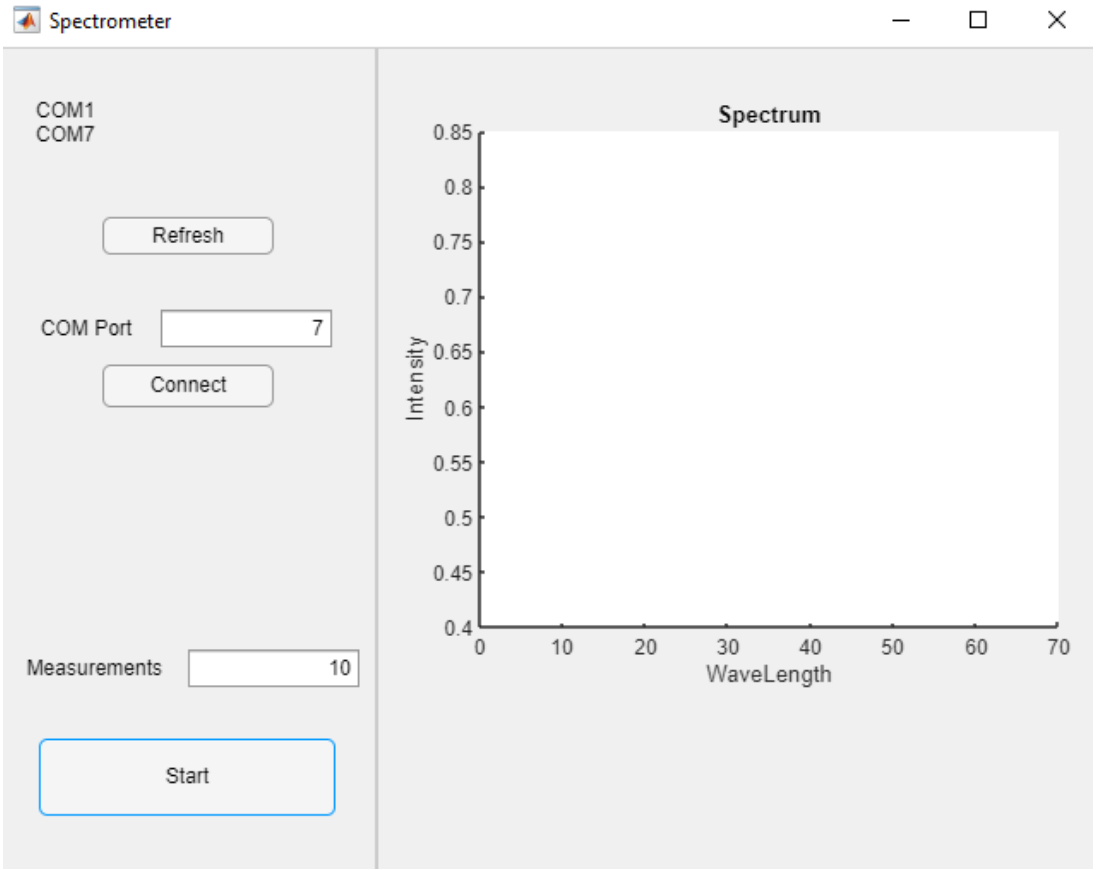


Figure 3: Interface finale

Le code Matlab de l'interface est le suivant :

```
classdef interface_fonctionnelle < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        SpectrometerUIFigure        matlab.ui.Figure
        GridLayout                  matlab.ui.container.GridLayout
        LeftPanel                   matlab.ui.container.Panel
        RefreshButton               matlab.ui.control.Button
        StartButton                 matlab.ui.control.Button
        MeasurementsEditField       matlab.ui.control.EditField
        MeasurementsEditFieldLabel  matlab.ui.control.Label
        PortListLabel               matlab.ui.control.Label
        COMPortEditField            matlab.ui.control.EditField
        COMPortEditFieldLabel       matlab.ui.control.Label
        ConnectButton               matlab.ui.control.Button
        RightPanel                  matlab.ui.container.Panel
        UIAxes                      matlab.ui.control.UIAxes
    end

    % Properties that correspond to apps with auto-reflow
    properties (Access = private)
        onePanelWidth = 576;
    end


    properties (Access = private)
        s % serial port
        portList % available comm ports
    end

    methods (Access = private)

        function rs232Interrupt(app, src, ~)
            rLine = readline(src);
            rLineChar = char(rLine);
            app.PortListLabel.Text = rLineChar;
            drawnow;
            flush(app.s)
        end

        function readpix(app, src, ~)
            data=readline(src);

            % Convert the string data to numeric type and save it in the UserData property
     of the serialport object.
            src.UserData.Data(end+1) = str2double(data);

            % Update the Count value of the serialport object.
            src.UserData.Count = src.UserData.Count + 1;

            if src.UserData.Count > 63
                configureCallback(src, "off");
            end
        end
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            app.portList = serialportlist();
            app.PortListLabel.Text = app.portList;
        end
```

```
65
66          % Button pushed function: ConnectButton
67          function ConnectButtonPushed(app, event)
68              % Connect the selected serial port
69              try
70                  app.s = serialport("COM"+app.COMPortEditField.Value, 115200);
71                  configureTerminator(app.s, "CR/LF");
72                  % Update the port list
73                  %configureCallback(app.s, "terminator", @app.rs232Interrupt);
74              catch
75                  warning('Problem using serial.');
76              end
77              drawnow;
78          end
79
80          % Button pushed function: StartButton
81          function StartButtonPushed(app, event)
82              %Read the number of times the spectrum will be measured
83              %configureTerminator(app.s, "CR/LF");
84              c=app.MeasurementsEditField.Value;
85              count=str2double(c);
86              sum=0;
87
88
89              %Perform measurements
90              for i=1:count
91              app.s.UserData = struct("Data",[],"Count",0);
92              configureCallback(app.s,"terminator",@app.readpix);
93              pause(1); %Time needed to acquire the data
94              sum = sum+app.s.UserData.Data;
95              end
96
97              %Find the average value and plot
98              aver=sum/count;
99              plot(app.UIAxes,1:64,aver);
100         end
101
102         % Button pushed function: RefreshButton
103         function RefreshButtonPushed(app, event)
104             app.portList = serialportlist();
105             app.PortListLabel.Text = app.portList;
106             drawnow;
107         end
108
109         % Changes arrangement of the app based on UIFigure width
110         function updateAppLayout(app, event)
111             currentFigureWidth = app.SpectrometerUIFigure.Position(3);
112             if(currentFigureWidth <= app.onePanelWidth)
113                 % Change to a 2x1 grid
114                 app.GridLayout.RowHeight = {480, 480};
115                 app.GridLayout.ColumnWidth = {'1x'};
116                 app.RightPanel.Layout.Row = 2;
117                 app.RightPanel.Layout.Column = 1;
118             else
119                 % Change to a 1x2 grid
120                 app.GridLayout.RowHeight = {'1x'};
121                 app.GridLayout.ColumnWidth = {220, '1x'};
122                 app.RightPanel.Layout.Row = 1;
123                 app.RightPanel.Layout.Column = 2;
124             end
125         end
126     end
127
128     % Component initialization
129     methods (Access = private)
130
```

```
        % Create UIFigure and components
        function createComponents(app)

            % Create SpectrometerUIFigure and hide until all components are created
            app.SpectrometerUIFigure = uifigure('Visible', 'off');
            app.SpectrometerUIFigure.AutoResizeChildren = 'off';
            app.SpectrometerUIFigure.Position = [100 100 640 480];
            app.SpectrometerUIFigure.Name = 'Spectrometer';
            app.SpectrometerUIFigure.SizeChangedFcn = createCallbackFcn(app,
    @updateAppLayout, true);

            % Create GridLayout
            app.GridLayout = uigridlayout(app.SpectrometerUIFigure);
            app.GridLayout.ColumnWidth = {220, '1x'};
            app.GridLayout.RowHeight = {'1x'};
            app.GridLayout.ColumnSpacing = 0;
            app.GridLayout.RowSpacing = 0;
            app.GridLayout.Padding = [0 0 0 0];
            app.GridLayout.Scrollable = 'on';

            % Create LeftPanel
            app.LeftPanel = uipanel(app.GridLayout);
            app.LeftPanel.Layout.Row = 1;
            app.LeftPanel.Layout.Column = 1;

            % Create ConnectButton
            app.ConnectButton = uibutton(app.LeftPanel, 'push');
            app.ConnectButton.ButtonPushedFcn = createCallbackFcn(app,
    @ConnectButtonPushed, true);
            app.ConnectButton.Position = [60 270 100 25];
            app.ConnectButton.Text = 'Connect';

            % Create COMPortEditFieldLabel
            app.COMPortEditFieldLabel = uilabel(app.LeftPanel);
            app.COMPortEditFieldLabel.HorizontalAlignment = 'center';
            app.COMPortEditFieldLabel.Position = [21 305 59 22];
            app.COMPortEditFieldLabel.Text = 'COM Port';

            % Create COMPortEditField
            app.COMPortEditField = uieditfield(app.LeftPanel, 'text');
            app.COMPortEditField.HorizontalAlignment = 'right';
            app.COMPortEditField.Position = [94 305 100 22];
            app.COMPortEditField.Value = '0';

            % Create PortListLabel
            app.PortListLabel = uilabel(app.LeftPanel);
            app.PortListLabel.Position = [21 401 50 70];
            app.PortListLabel.Text = 'Port List';

            % Create MeasurementsEditFieldLabel
            app.MeasurementsEditFieldLabel = uilabel(app.LeftPanel);
            app.MeasurementsEditFieldLabel.HorizontalAlignment = 'right';
            app.MeasurementsEditFieldLabel.Position = [10 107 85 22];
            app.MeasurementsEditFieldLabel.Text = 'Measurements';

            % Create MeasurementsEditField
            app.MeasurementsEditField = uieditfield(app.LeftPanel, 'text');
            app.MeasurementsEditField.HorizontalAlignment = 'right';
            app.MeasurementsEditField.Position = [110 107 100 22];
            app.MeasurementsEditField.Value = '10';

            % Create StartButton
            app.StartButton = uibutton(app.LeftPanel, 'push');
            app.StartButton.ButtonPushedFcn = createCallbackFcn(app, @StartButtonPushed,
    true);
            app.StartButton.Position = [23 32 173 45];
```

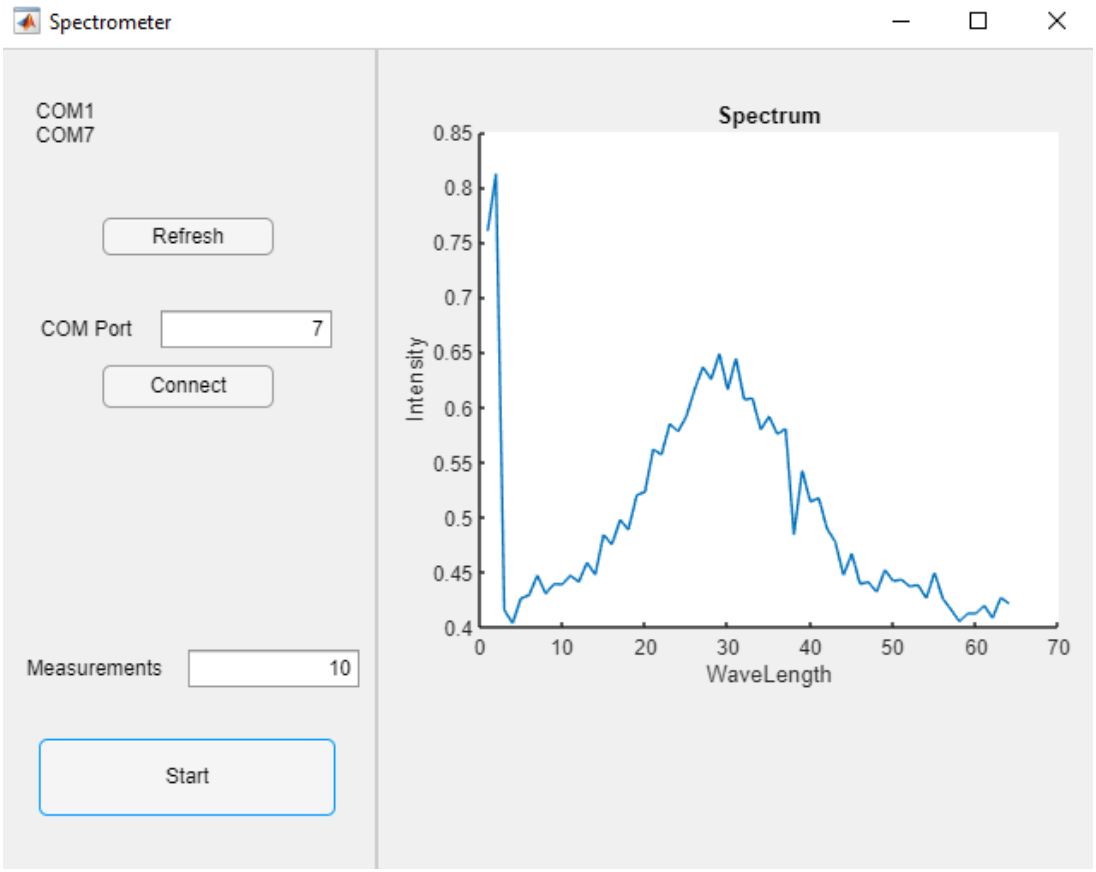
```
            app.StartButton.Text = 'Start';

            % Create RefreshButton
            app.RefreshButton = uibutton(app.LeftPanel, 'push');
            app.RefreshButton.ButtonPushedFcn = createCallbackFcn(app,
    @RefreshButtonPushed, true);
            app.RefreshButton.Position = [60 359 100 22];
            app.RefreshButton.Text = 'Refresh';

            % Create RightPanel
            app.RightPanel = uipanel(app.GridLayout);
            app.RightPanel.Layout.Row = 1;
            app.RightPanel.Layout.Column = 2;

            % Create UIAxes
            app.UIAxes = uiaxes(app.RightPanel);
            title(app.UIAxes, 'Spectrum')
            xlabel(app.UIAxes, 'WaveLength')
            ylabel(app.UIAxes, 'Intensity')
            zlabel(app.UIAxes, 'Z')
            app.UIAxes.Position = [14 107 391 343];

            % Show the figure after all components are created
            app.SpectrometerUIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = interface_fonctionnelle

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.SpectrometerUIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.SpectrometerUIFigure)
        end
    end
end
```

4 Tests du système et confrontation avec les objectifs initiaux

Une fois les différents blocs du système combinés, on a pu tester notre spectromètre sur la lampe spectrale à mercure que nous avions à disposition.

Là où nous souhaitions initialement observer l'ordre 1, celui-ci ne présentant aucun problème de recouvrement d'ordre, nous avons constaté qu'il n'était pas assez brillant pour être observé avec le capteur CCD. Nous avons donc été contraints d'observer l'**ordre 5**, plus brillant, pour obtenir des signaux satisfaisants en sortie. On essaye alors d'observer et de distinguer une raie bleu/violette et une raie verte, proches mais tout de même suffisament éloignées pour être distinguables.

Voici ce qu'on observe sur l'interface finale :



On constate qu'on ne distingue ici qu'une seule raie, qui est le résultat de la fusion des deux raies qu'on souhaitait observer. La résolution de notre système n'est visiblement pas assez bonne, de l'ordre de **20 nm**.

On peut tout de même remarquer que nous avons réussi à faire communiquer le capteur CCD et l'ordinateur grâce à une carte Nucléo, pour ensuite utiliser les données récupérées dans une interface. Ainsi les objectifs touchant au numérique que nous nous étions fixés ont été réalisés.

5 Ouverture et points d'amélioration

On a constaté dans la partie précédente que les objectifs que nous nous étions fixés n'ont pas tous été atteints. Une façon d'améliorer la résolution de notre spectromètre serait de changer le réseau utilisé : avec un réseau de pas plus petit, on pourrait mieux écarter les différentes raies, même si cela implique aussi que l'on observerait une *plus petite partie* du spectre sur notre capteur.

Il serait aussi judicieux de prendre en compte la sensibilité variable de la barette CCD selon la longueur d'onde dans le code Matlab.

Enfin, il serait aussi possible de prendre un capteur plus performant, pour avoir des mesures plus précises.

6 Conclusion

Ce projet aura été l'occasion de travailler en équipe sur un sujet mêlant optique et numérique.
Malgré des problèmes survenus au cours du projet, dus notamment au capteur CCD et au goniomètre utilisé, nous avons pu tirer divers enseignements et constats de ces nombreuses séances. Le fait que nous nous connaissions déjà et nous entendions bien nous a permis de facilement nous répartir le travail, et de communiquer au sein du groupe sans difficultés notables. On a aussi pu constater l'importance de bien prendre en note tout ce qui est accompli lors des séances sur un carnet de bord commun, tant les séances ont pu être écartées temporellement les unes des autres.
Enfin, sur une note plus légère, on se souviendra de ce qu'on nous disait en début de projet : même si tout ne marche pas forcément, il y a beaucoup à retenir d'un tel projet, notament au niveau des compétences utiles au travail de groupe.